VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Katedra informatiky

Implementace složitějších kódů proměnné délky
Implementation of Advanced Variable-length Codes

2011 **Radek Kuzník**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radek Kuzník**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace složitějších kódů proměnné délky
Implementation of Advanced Variable-length Codes

Zásady pro vypracování:

Cílem práce je nastudovat a implementovat několik pokročilejších typů kódování pro symboly/čísla, porovnat je z hlediska efektivity a hlavně připravit je pro použití v kompresním frameworku.

Postupujte dle následujících bodů:
1. Nastudujte následující typy kódování:
   - Eliasovy kódy Gamma, Delta a Omega,
   - Even-Rodeh kódy,
   - Punctured Eliasovy kódy,
   - Stoutovy kódy.
2. Implementujte jednotlivá kódování.
3. Otestujte jejich efektivity pro různé druhy dat.
4. Sestavte moduly pro jednotlivá kódování pro použití v kompresním frameworku.

Seznam doporučené odborné literatury:

Variable-length codes for Data Compression, David Salomon, Springer, 2007
Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 3. května 2011 …….

podpis

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Janu Platošovi za pomoc a rady při tvorbě této bakalářské práce.

Dále děkuji své rodině za psychickou podporu a porozumění, které mi velice pomáhalo nejen v době realizace bakalářské práce, ale po celou dobu studií.

Abstrakt

Cílem práce bylo nastudovat několik pokročilejších typů kódování pro symboly/čísla, provést jejich implementaci, otestovat je z hlediska efektivity a připravit je jako moduly pro použití v kompresním frameworku. Po konzultaci s vedoucím práce bylo od implementace Stoutova S_l kódu upuštěno. V této práci byly použity Eliasovy kódy Gama, Delta a Omega, Even-Rodeh kódy, Punctured Eliasovy kódy a Stoutův R_l kód. Získané znalosti o těchto kódech byly následně použity při tvorbě modulů a testování. Testování efektivity bylo provedeno na náhodných celých číslech v rozmezí < 1,2147483647 >, které obsahovaly poskytnuté testovací soubory.

Klíčová slova

Eliasův gama kód, Eliasův delta kód, Eliasův omega kód, Even-Rodeh kód, Punctured Eliasův kód, Stoutův kód, komprese dat

Abstract

The aim of work was to study a few types of advanced encoding for symbols / numbers to implement them, test them in terms of efficiency and to prepare them as modules for use in the compression framework. After consultation with the supervisor was implementation of Stout S_l code canceled. In this work are used Elias Gamma, Delta and Omega codes, Even-Rodeh codes, Punctured Elias codes and Stout R_l code. Knowledge of these codes were used to create modules and testing. Testing of the effectiveness was performed on random integers in the range <1 2147483647>, which contained the test files.

Keywords

Elias gamma code, Elias delta code, Elias omega code, Even-Rodeh code, Punctured Elias code, Stout code, data compression

Seznam použitých zkratek a symbolů

α – unární kód, sekvence nul ukončená jedničkou

β – beta kód, klasický bitový zápis

Obsah

1. Úvod

Stále více lidí používá internet jako zdroj informací, zábavy a elektronického obchodování. To vede k větším nárokům na kapacitu přenosu dat a úložišť. Bohužel navyšování těchto kapacit je poměrně nákladná záležitost, bylo tedy potřeba vymyslet vhodnou metodu, která by tento problém vyřešila i bez zásahu do fyzického návrhu. To vedlo ke vzniku různých metod zmenšujících objem dat, čímž se celkový přenos dat zrychlil a dostalo se tak více prostoru pro další data. Tento speciální postup se nazývá komprimace (komprese) dat.

Hlavním úkolem komprimace dat je zmenšení objemu dat odstraněním nadbytečných informací před jejich uložením nebo přenosem. Dosahuje se toho různými komprimačními postupy a musí obsahovat i opačný postup pro dekomprimaci, který původní informaci opět zrekonstruuje.

Komprimaci dat lze rozdělit na dvě základní kategorie. První kategorie tzv. ztrátová komprimace odstraňuje nenávratně části informace do té míry, dokud ji lze uspokojivě zrekonstruovat. Spoléhá především na nedokonalost lidských smyslů, kdy si člověk chybějící části nevšimne nebo si jí dokáže domyslet. U druhé kategorie bezeztrátové komprimace je celá informace zachována a lze ji zpětně zrekonstruovat do předchozího stavu bez jakékoliv ztráty dat. Toto je nutná podmínka při přenosu, kde by ztráta jediného bitu znamenala ztrátu celé informace. Úspěšnost této komprimace závisí především na typu dat a obvykle není tak účinná jako ztrátová komprimace dat.

Aby bylo možné jednotlivé komprimační metody porovnat, používá se pro jejich metriku tzv. komprimační poměr, který se vypočítá jako podíl velikosti původních dat k velikosti zkomprimovaných dat a nejčastěji se vyjadřuje v procentech. Další metodou pro měření je faktor komprimace, jedná se o převrácenou hodnotu komprimačního poměru.

Úkolem této práce je použití typů kódování, které zaručují bezeztrátovou komprimaci. Nové moduly budou implementovány v programovacím jazyku C++ a budou kompatibilní se zbytkem systému.

Text bakalářské práce je rozdělen do několika zájmových celků. Nejprve se seznámíme s několika typy komprimačních kódování a jejich metodami pro komprimaci/dekomprimaci. Dalším zájmovým celkem je samotná implementace jednotlivých modulů, kde se objevuje i část zdrojových kódů. Podstatná část se pak věnuje samotnému testování těchto kódování z hlediska efektivity. A závěr shrnuje úspěšnost jednotlivých kódování.

2. Prefixové kódy

Všechny následující použité kódy jsou tzv. prefixové kódy, jejichž základní vlastností je, že žádný prefix jednoho symbolu není kódem jiného symbolu. Pokud je kód prefixový, lze řetězce symbolů tohoto kódu jednoznačně dekódovat, aniž by byly použity oddělovače mezi jednotlivými symboly. Velkou výhodou prefixových kódu je možnost text snadno dekódovat zleva doprava. Celý princip funguje tak, že se odebírají bity tak dlouho, dokud nedostaneme kód některého symbolu, čímž je symbol dekódován. Tímto způsobem dekódujeme jeden symbol za druhým.

2.1 Eliasovy kódy

Autor těchto kódů Peter Elias, byl americký průkopník dvacátého století v oblasti teoretické informatiky. Elias popsal v roce 1975 tři užitečné prefixové kódy a pojmenoval je Gama (γ), Delta (δ), Omega (ω).

Jeho hlavní myšlenkou bylo vytvořit prefix z velikosti jejich délky. Vymyslel tedy různé způsoby jak je zapisovat. Elias popsal unární kód čísla *n* jako $\alpha(n)$ a klasický bitový zápis jako $\beta(n)$.

2.2 Eliasův gama kód

Eliasův gama kód je jednoduchý algoritmus, vytvořený pro kladná celá čísla. Používá se pro komprimaci dat, kde se předpokládá, že malá čísla se objevují častěji než velká a tam, kde není předem známa velikost největší zakódované hodnoty. V tomto typu kódování je použit alfa a beta kód.

Výsledný gama (γ) kód je dán vztahem $\gamma(n) = \alpha(|\beta(n)|) + \beta(L)$.

Kódování

1. Zakódujeme číslo *n* kódem beta $\beta(n)$.
2. Spočítáme délku tohoto kódu, $M = |\beta(n)|$.
3. Zakódujeme číslo *M* unárním kódem alfa, $\alpha(M)$.
4. Zapíšeme kód $\alpha(M)$ následovaný kódem $\beta(n)$ bez nejvyšší jedničky.

Délku *M* čísla *n* získáme z rovnice pro výpočet délky beta kódu, $1 + \lfloor \log_2 n \rfloor$. Výsledná velikost gama (γ) kódu pro číslo *n* je

$$2M - 1 = 2\lfloor \log_2 n \rfloor.$$

$1 = 2^0 + 0 =$ **1**	$10 = 2^3 + 2 =$ **0001***010*
$2 = 2^1 + 0 =$ **01***0*	$11 = 2^3 + 3 =$ **0001***011*
$3 = 2^1 + 1 =$ **01***1*	$12 = 2^3 + 4 =$ **0001***100*
$4 = 2^2 + 0 =$ **001***00*	$13 = 2^3 + 5 =$ **0001***101*
$5 = 2^2 + 1 =$ **001***01*	$14 = 2^3 + 6 =$ **0001***110*
$6 = 2^2 + 2 =$ **001***10*	$15 = 2^3 + 7 =$ **0001***111*
$7 = 2^2 + 3 =$ **001***11*	$16 = 2^4 + 0 =$ **00001***0000*
$8 = 2^3 + 0 =$ **0001***000*	$17 = 2^4 + 1 =$ **00001***0001*
$9 = 2^3 + 1 =$ **0001***001*	$18 = 2^4 + 2 =$ **00001***0010*

Ukázka 2.2.1: Eliasovo gama kódování

Dekódování

1. Čteme nuly, dokud nenačteme jedničku a zapíšeme počet načtených nul jako *N*.
2. Čteme dalších *N* bitů a označíme je jako *L*.
3. Vypočítáme $n = 2^N + L$.

2.3 Eliasův delta kód

Na rozdíl od předešlého kódování je délka kódu u Eliasova delta kódu zapsána beta kódem (β). Tvorba je o trochu složitější než u gama kódu a jedná se o univerzální prefixový kód pro kladná celá čísla.

Kódování

1. Zakódujeme číslo *n* kódem beta $\beta(n)$.
2. Spočítáme délku tohoto kódu, $M = |\beta(n)|$.
3. Zakódujeme číslo *M* kódem beta $\beta(M)$.
4. Spočítáme délku kódu z kroku 3, $C = |\beta(M)|$.
5. Zapíšeme *C* - 1 nul následované kódem beta $\beta(M)$ a kódem beta $\beta(n)$ bez nejvyšší jedničky.

1 = 1	10 = 00100010
2 = 0100	11 = 00100011
3 = 0101	12 = 00100100
4 = 01100	13 = 00100101
5 = 01101	14 = 00100110
6 = 01110	15 = 00100111
7 = 01111	16 = 001010000
8 = 00100000	17 = 001010001
9 = 00100001	18 = 001010010

Ukázka 2.3.1: Eliasovo delta kódování

Pro výpočet délky delta kódování čísla n, upravíme rovnici pro délku M z prvního kroku delta kódování, $M = 1 + \lfloor \log_2 n \rfloor$. Upravíme rovnici následovně

$$M = 1 + \log_2 n = \log_2 2 + \log_2 n = \log_2(2n).$$

V kroku 4 jsme počítali velikost beta kódu pro číslo M, kde délka N je poté dána vztahem

$$N = 1 + \log_2 M = 1 + \log_2(\log_2(2n))$$

V posledním kroku odčítáme číslo jedna z N a délka tohoto kódu je dána jako

$$N - 1 = \log_2 M = \log_2(\log_2(2n)).$$

Celková délka delta kódu je dána součtem těchto tří rovnic, kde nezapomeneme odečíst jedničku z druhé rovnice, která vznikla při odečtení nejvyšší jedničky.

$$\log_2(2n) + [1 + \log_2 \log_2(2n)] - 1 + \log_2 \log_2(2n) = 1 + \lfloor \log_2 n \rfloor + 2\lfloor \log_2 \log_2(2n) \rfloor$$

Dekódování

1. Čteme nulové bity, dokud nenarazíme na jedničku a zapíšeme počet nul jako N.
2. Čteme dalších N+1 bitů jako celé číslo a zapíšeme jako M.
3. Čteme posledních M - 1 bitů,
4. Přidáme jedničku před kód z kroku 3a převedeme na celé číslo.

2.4 Eliasův omega kód

Poslední Eliasův kód je omega kód, který je někdy také označován jako Eliasův rekurzivní kód, jedná se také o univerzální prefixový kód pro kladná celá čísla a je vhodný pro velmi vysoká čísla. Na rozdíl od jeho předchozích kódů využívá omega kód rekurzivní kódování prefixu z délky beta kódování. Hlavní myšlenka tohoto kódování je, že předcházející skupina reprezentuje délku následující skupiny zmenšenou o jedna, kde první skupina má délku 2. Pro rozlišení zakódovaných čísel se na konec kódování přidává oddělovač 0 a u tohoto kódování můžeme narazit i na speciální případ, kdy se číslo $n = 1$.

Kódování

1. Zapíšeme číslo 0.
2. Pokud číslo n není rovno číslu 1, zakódujeme číslo n kódem beta $\beta(n)$ a zapíšeme délku, $M = |\beta(n)|$. Pokud je číslo n rovno 1 ukončíme kódování.
3. Opakujeme druhý krok pro $n = M - 1$.

1 = 0	10 = 11 1010 0
2 = 10 0	11 = 11 1011 0
3 = 11 0	12 = 11 1100 0
4 = 10 100 0	13 = 11 1101 0
5 = 10 101 0	14 = 11 1110 0
6 = 10 110 0	15 = 11 1111 0
7 = 10 111 0	16 = 10 100 10000 0
8 = 11 1000 0	17 = 10 100 10001 0
9 = 11 1001 0	18 = 10 100 10010 0

Ukázka 2.4.1: Eliasovo omega kódování

Při prvním pohledu na tabulku Eliasových omega kódu si můžeme, všimnou pomalého růstu délky kódování s tím, jak se číslo *n* zvětšuje. Ale pokud dojde ke vzniku nové skupiny, délka kódování se náhle zvětší o několik bitů. Tento děj nastává v případech kdy číslo $n = 2^{2^k}$ a k je kladné celé číslo. Pro hodnoty *k* rovny 1, 2, 3 a 4, se toto stane, když číslo *n* dosáhne hodnot 4, 16, 256 a 65536.

Protože Eliasovo omega kódování je rekurzivní, je i jeho délka počítána rekurzivně. Celková velikost omega kódování je dána jako posloupnost beta kódu a je dána vztahem

$$|\omega(n)| = \sum_{i=1}^{k} \beta\left(l^{k-i}(n)\right) + 1 = 1 + \sum_{i=1}^{k} (l^i(n) + 1),$$

kde je součet ukončen při splnění podmínky $l^k(n) = 1$ pro číslo *k*.

Dekódování

Provede se několika kroky, kde každý krok načte skupinu bitů z kódu. Pokud skupina začíná nulou je dekódování ukončeno.

1. Zapíšeme *n* = 1.
2. Načteme první skupinu, která obsahuje jeden bit 0 nebo bit 1 následovaný dalšími *n* bity. Pokud je bit 0, je výsledné číslo 1, jinak převedeme skupinu bitů na celé číslo a zapíšeme jako *n*.
3. Načteme další skupinu, která obsahuje jeden bit 0 nebo bit 1 následovaný dalšími *n* bity. Pokud je bit 0, je výsledné číslo rovno *n* a dekódování se ukončí. Když je bit 1, převedeme skupinu bitů na celé číslo, zapíšeme je jako *n* a opakujeme tento krok.

2.5 Even-Rodeh kód

Vznikl v roce 1978 a jeho autory jsou Shimon Even a Michael Rodeh. Kód je velmi podobný Eliasovu omega kódu a liší se při kódování do délky 3 bitů, kde se u Even-Rodeh kódu přidávají prefixové nuly u první skupiny při délce do 3 bitů.

Kódování

1. Zakódujeme číslo n kódem beta $\beta(n)$ a spočítáme délku tohoto kódu, $M = |\beta(n)|$.
2. Pokud je číslo M menší než 3, přidáme $3 - M$ nul před kód a ukončíme kódování.
3. Je-li číslo M rovno 3, ukončíme kódování a přidáme na konec kódu nulu. Pokud je M větší než 3 zakódujeme číslo M beta kódem $\beta(M)$ a zapíšeme délku, $L = |\beta(M)|$.
4. Krok 3 opakujeme pro $M = L$.

0 = 000	15 = 100 1111 0
1 = 001	16 = 101 10000 0
2 = 010	32 = 110 100000 0
3 = 011	100 = 111 1100100 0
4 = 100 0	1000 = 100 1010 1111101000 0
5 = 101 0	2761 = 100 1100 101011001001 0
6 = 110 0	10000 = 100 1110 10011100010000 0
7 = 111 0	50000 = 101 10000 1100001101010000 0
8 = 100 1000 0	100000 = 101 10001 11000011010100000 0

Ukázka 2.5.1: Even-Rodeh kódování

Dekódování

1. Načteme první tři bity jako celé číslo n. Je-li první bit v této skupině roven 0, ukončíme dekódování, jinak pokračujeme krokem 2.
2. Načteme další skupinu, která obsahuje jeden bit 0 nebo bit 1 následovaný dalšími bity. Pokud je bit 0, pak je výsledné číslo n a dekódování končí. Pokud je bit 1, převedeme skupinu bitů na celé číslo a zapíšeme jako n.
3. Opakujeme krok 2.

2.6 Punctured Eliasovy kódy

Při pokusu vylepšit výkon Burrow-Wheeler transformace objevil Peter Fenwick v roce 1996 dva typy kódování. Tyto druhy kódů označili jako P1 a P2, kde P2 je odvozený z P1. Termín punctured vzešel z oblasti kontroly chyb v kódech. Nejčastěji data obsahují spolu s vlastní informací i doplňkovou informaci, která slouží k ověření informace, zda při jejím přenosu nedošlo k chybě. Pokud jsou některé kontrolní bity odstraněny kvůli zkracování kódu, je výsledný kód nazýván jako Punctured.

Kódování P1

1. Zakódujeme číslo n kódem beta $\beta(n)$ a zapíšeme počet jedniček jako N.
2. Obrátíme pořadí bitů $\beta(n)$.
3. Přidáme před beta kód N jedniček a nulu.

Kódování P2

1. Číslo n zvýšíme o 1.
2. Zakódujeme číslo n kódem beta $\beta(n)$ a zapíšeme počet jedniček jako N.
3. Obrátíme pořadí bitů $\beta(n)$.
4. Přidáme před beta kód N - 1 jedniček a nulu.

n	P1	P2
0	0	01
1	101	001
2	1001	1011
3	11011	0001
4	10001	10101
5	110101	10011
6	110011	110111
7	1110111	00001
8	100001	101001
9	1101001	100101
10	1100101	1101101

Ukázka 2.6.1: Punctured Elias kódování

Z tabulky 1.5 si můžeme všimnout, že délka kódu se zvětšuje s rostoucím číslem n. Avšak, pokud je číslo n rovno mocnině dvou, jeho délka se zmenší.

Dekódování P1

1. Načítáme jedničky dokud nenarazíme na bit nula a zapíšeme tento počet jako *N*.
2. Nulový bit z kroku jedna přeskočíme a načítáme bity, dokud počet načtených jedniček není roven *N*.
3. Převrátíme pořadí bitu z kroku 2 a převedeme na celé číslo.

Dekódování P2

1. Načítáme jedničky dokud nenarazíme na bit nula a zapíšeme tento počet jako *N*.
2. Nulový bit z kroku jedna přeskočíme a načítáme bity, dokud počet načtených jedniček není roven $N + 1$.
4. Převrátíme pořadí bitu z kroku 2 a převedeme na celé číslo.
3. Získané číslo z kroku 3 snížíme o 1.

2.7 Stoutův R_l kód

V roce 1980 Quentin Stout představil rekurzivní kódování pojmenované R_l pro celá čísla, podobné a mnohem obecnější než Elias omega a Even-Rodeh kódování. Kód je univerzální a velikost délky beta kódování je dána vztahem $L = 1 + \lfloor \log_2 n \rfloor$.

Toto kódování je závislé na výběru hodnoty parametru celého čísla l a může nabývat hodnoty dva a více. Stoutovo kódování R_l se skládá z prefixu $R_l(n)$ a sufixu $0n$. Suffix je binární hodnota n v L bitech a předchází jej oddělovač číslo 0. Prefix $R_l(n)$ se skládá z délek skupin. Začíná délkou L čísla n, které předchází délka L_1 z L, dál předchází délka L_2 z L_1 , dokud se délka L_i nerovná parametru l. Můžeme si všimnout, že délky skupin (s výjimkou první levé skupiny) začínají 1 a proto oddělovač 0 značí konec délek skupin.

Kódování

1. Zakódujeme číslo n kódem beta $\beta(n)$ a spočítáme délku tohoto kódu, $M = |\beta(n)|$.
2. Přidáme oddělovač 0 před kód beta $\beta(n)$.
3. Pokud je číslo M menší nebo rovno než l, ukončíme kódování, a při M menší než l, přidáme ještě $l - M$ nul před kód beta.
4. Je-li číslo M větší než číslo l, opakujeme od kroku 1(bez kroku 2) a dosadíme $n = M$.

Nejkratší prefixy získáme, když parametr $l = L = 1 + \lfloor \log_2 n \rfloor$. Při větších hodnotách l je výsledný prefix o trochu delší a u malých hodnot přibývá počet skupin délek a to má za následek mnohem delší prefix. Rozsah nejlepších délek L je pro daný parametr l dán rovnicí $[2^s, 2^e - 1]$ kde $s = 2^{l-1}$ a $e = 2^l - 1 = 2s - 1$. Tabulka 2.1.7 ukazuje nejlepší hodnotu parametru l pro dané rozsahy.

$R_2(985)$ =	11 100 1010 1111011001	$R_2(31\,925)$ =	11 100 1111 111110010110101
$R_3(985)$ =	100 1010 1111011001	$R_3(31\,925)$ =	100 1111 111110010110101
$R_4(985)$ =	1010 1111011001	$R_4(31\,925)$ =	1111 111110010110101
$R_5(985)$ =	01010 1111011001	$R_5(31\,925)$ =	01111 111110010110101
$R_6(985)$ =	001010 1111011001	$R_6(31\,925)$ =	001111 111110010110101

Ukázka 2.7.1: Prefixy čísel 985 a 31925 pro Stoutovo R_l kódování

Dekódování

Dekódování je poměrně jednoduché, dekodér načte prvních l bitů, které značí délku další skupiny a takto se pokračuje, dokud nalezená skupina nezačíná 0, což je oddělovač, který indikuje skupinu čísla n.

l	s	e	2^s	$2^e - 1$
2	2	3	2	7
3	4	7	8	127
4	8	15	128	32 767
5	9	31	32 768	2 147 483 647

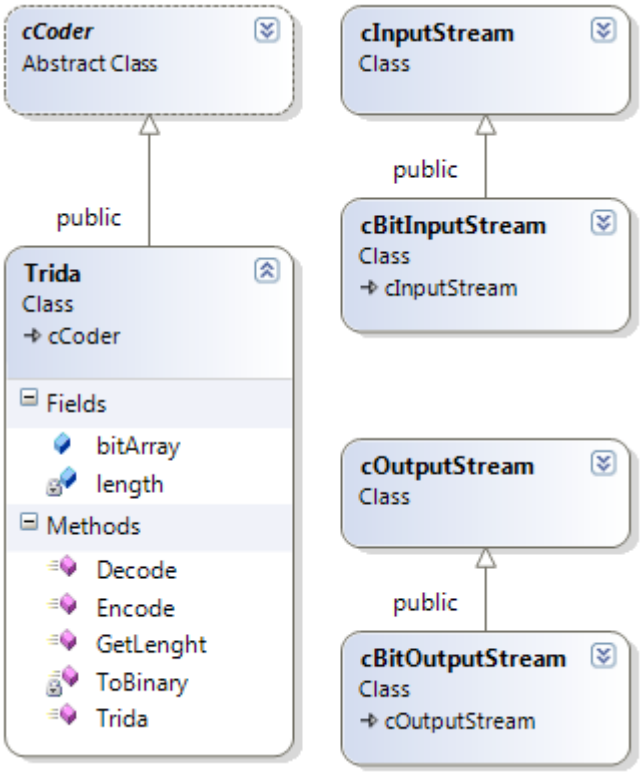
Tabulka 2.7.1: Nejlepší hodnota l pro dané rozsahy

3. Implementace

Implementace probíhala v programovacím jazyce C++ spolu s hlavičkovovými soubory a třídami cBitInputStream, cBitOutputStream, cInputStream, cOutputStream, cCoder, které byly vytvořené vedoucím práce.

Mnou vytvořené třídy a hlavičkové soubory byly pojmenovány podle anglického názvu kódování (Elias gamma, Elias delta, Elias omega, Even-Rodeh, Punctured Elias a Stout) a dědí z abstraktní třídy cCoder. Třída cCoder obsahuje metody pro kódování, dekódování a výpočet pro funkci $\log_2$.Tyto metody jsem použil při tvorbě vlastních tříd. Všechny moje třídy obsahují funkci int *Encode*(int *n*) pro kódování čísla *n*, funkci int *Decode()* pro dekódování, funkci *GetLength()* pro zjištění délky kódu a funkci *ToBinary*(int *n*) pro kódování beta čísla *n*.

Na Obrázku č. 3.1 můžeme vidět zjednodušený diagram tříd. Pro ukázku jsem vytvořil obecnou třídu Trida, která obsahuje metody společné pro všechny mé vytvořené třídy.



Obrázek 3.1: Diagram tříd

Z důvodu opačného pořadí v jakém se bity ukládají do streamu, jsem vytvořil proměnné *pos* a *len*, a pomocné pole celých čísel *bitArray* pro ukládání bitu. Každý zápis do pole *bitArray* zvyšuje proměnnou *pos* o jedna a značí aktuální pozici v tomto poli. Proměnná *len,* obsahuje velikost beta kódu čísla *n*, kterou vrací funkce *ToBinary()*. Pro lepší znázornění jsou uvedeny příklady kódování/dekódování.

3.1 Implementace Eliasova gama kódování

Jedná se o nejjednodušší kódování. Implementaci Eliasova kódování jsem provedl s malým rozdílem oproti původnímu postupu kódování. Má implementace ponechá nejvyšší jedničku a přidá pouze *M*-1 nul namísto toho aby byla nejvyšší jednička smazána a vložen unární kód.

Problematiku tohoto kódování jsem rozdělil do dvou cyklů. První cyklus ukládá beta kód čísla *n,* $\beta(n)$ od konce pole *bitArray*. Jakmile je beta kódování dokončeno, začíná další cyklus, který vkládá nuly od následující pozice.

U dekódování se načítají počáteční nulové bity ze streamu do proměnné *countZero*. Pokud je načtena jednička, začíná cyklus, který načte dalších *countZero* + 1 bitů a dekóduje bity na celé číslo spolu s prvním načteným jedničkovým bitem.

Příklad 3.1.1
Zvolíme $n = 18$. V prvním kroku číslo *n* zakódujeme kódem beta, $18_{10} = 10010_2$ a $len = 5$. V dalším kroku vkládáme *len* - 1 nul od následující pozice, čímž je celé kódování dokončeno. Výsledný gama kód má tvar 0000|10010.

Příklad 3.1.2
Postup při dekódování kódu 000010011_2 je následující. Ze streamu načítáme nuly, dokud nenarazíme na jedničku a počet načtených nul zapíšeme, *countZero* = 4. Nyní víme, že je třeba načíst dalších *countZero* + 1 bitů a dané bity převést, odkódované číslo je 19.

3.2 Implementace Eliasova delta kódování

Na rozdíl od Eliasova gama kódování dochází při delta kódování k dvěma beta kódováním. U původní funkce *ToBinary()* byl pevně dán začátek zápisu do pole *bitArray* nešlo zapisovat od libovolné pozice. Přidal jsem parametr, který značí začátek zápisu od libovolného místa.

Kódování čísla probíhá následovně. V první smyčce zakódujeme číslo n kódem beta $\beta(n)$, který ukládáme od konce pole *bitArray*. V druhém kroku zakódujeme číslo *len* kódem beta $\beta(len)$ a zapisujeme tento kód od nejvyšší jedničky $\beta(n)$. V posledním kroku zapisujeme nuly od následující pozice.

Při dekódování se načítají počáteční nulové bity ze streamu do proměnné *countZero*. Pokud je načtena jednička, začíná cyklus, který načte dalších *countZero* + 1 bitů. Bity dekóduje na celé číslo a zapíše do proměnné *M*. Na závěr se načte *M* - 1 bitů, kde se přidá nejvyšší jednička, a všechny tyto bity se převedou na celé číslo.

Příklad 3.2.1
Zvolíme $n = 18$. V prvním kroku číslo n zakódujeme kódem beta, $18_{10} = 10010_2$ a $len = 5$. V dalším kroku zakódujeme kódem beta číslo *len*, $5_{10} = 101_2$, $len = 3$ a nyní vkládáme od poslední pozice, čímž se přepíše nejvyšší jednička $\beta(n)$ $101|0010_2$. Na závěr přidáme *len* - 1 nul od následující pozice a celé kódování je dokončeno, $00|101|0010_2$.

Příklad 3.2.2
Postup při dekódování delta kódu 001010011_2 je následující. Ze streamu načítáme nuly, dokud nenarazíme na jedničku a počet načtených nul zapíšeme, *countZero* = 2. Nyní víme, že je třeba načíst dalších *countZero* + 1 bitů, 101_2 a bity převedeme na číslo, $M = 5$. Načteme posledních $M - 1$ bitů jako celé číslo s jedničkou jako nejvyšší bit a získáme číslo 19.

3.3 Implementace Eliasova omega kódování

Jak je známo Eliasův omega kód je rekurzivní, proto jsem vytvořil metody recursiveEncode a recursiveDecode pro rekurzivní kódování/dekódování. Ošetřen je i speciální případ, který nastane, pokud je číslo $n = 1$.

```
void eliasOmega :: recursiveEncode(int l, int &pos)//l je delká β(n)
{
	l -= 1;
	if(l != 1)
	{
		int len = 0;
		len = ToBinary(l, pos);
		recursiveEncode(len, pos);
	}
}
```

Obrázek 3.3.1: Metoda eliasOmega::recursiveEncode(int l, int &pos)

U kódování se provedou následující kroky. Provede se kontrola, zda číslo n se rovná 1, pokud ano zapíše se do streamu nula a kódování je ukončeno. Pokud ne, na řadě je další krok, který zakóduje číslo n kódem beta $\beta(n)$ a ukládá je od oddělovače 0 v poli *bitArray*. V třetím kroku přichází na řadu metoda recursiveEncode, která délku l sníží o jedna a zjistí, zda není rovné 1. Pokud ano, je metoda ukončena. Pokud ne, je číslo zakódováno kódem beta a je opět volána metoda recursiveEncode s novým l a *pos*.

Dekódování začíná načtením jednoho bitu a provede se kontrola, zda je bit jednička nebo nula. Pokud je bit nulový, je dekódování ukončeno a číslo $n = 1$. Pokud ne, načte se další bit a převede se i s prvním jedničkovým bitem na celé číslo M. Dále se načte další bit a opět se provede kontrola, zda je bit nula či jednička. Pokud je bit 0, je výsledné číslo rovno číslu M. Pokud je bit 1, přichází na řadu metoda recursiveDecode, která načte dalších M bitů a převede je s prvním jedničkovým bitem na celé číslo M. Poté je metoda recursiveDecode volána tak dlouho dokud není první bit nula.

Příklad 3.3.1
Kódování čísla $n = 18$. Zapíšeme číslo 0. V druhém kroku číslo n zakódujeme kódem beta, $18_{10} = 10010_2$ a $len = 5$. V dalším kroku odečteme číslo jedna z len a zakódujeme beta kódem, $4_{10} = 100_2$, $len = 3$. Nakonec opět odečteme číslo jedna z len, a zakódujeme beta kódem, $2_{10} = 10_2$, $len = 2$. Číslo len se při opětovném snížení o jedna rovná jedné a kódování je ukončeno. Výsledný omega kód je $10|100|10010|0_2$.

Příklad 3.3.2
Postup při dekódování omega kódu 10100100110_2 je následující. Ze streamu načteme první 2 bity, 10_2. Převedeme je na celé číslo, $M = 2$. Načteme dalších $M + 1$ bitů jako celé číslo, $M = 4$. Načteme posledních $M + 1$ bitů jako celé číslo, $M = 18$. Další skupina bitů by začínala oddělovačem 0, který značí ukončení dekódování.

3.4 Implementace Even-Rodeh kódování

Toto kódování má tu zvláštnost, že pokud je číslo menší než 4, přidají se před kód nuly do délky 3 bitů.

U kódování se v prvním kroku zjistí, zda je číslo n menší než 4. Pokud ano, je číslo zakódováno beta kódem, přidáno $3 - len$ nul před kód a kódování je ukončeno. Pokud ne, zakódujeme číslo n kódem beta a voláme metodu recursiveEncode, která zkontroluje, zda délka není 3, pokud ano je metoda ukončena. Pokud ne je délka zakódovaná beta kódem a je volána opět metoda recursiveEncode. Takto to pokračuje do té doby, dokud není délka kódu 3.

Při dekódování je načten jeden bit. Pokud je bit 0, načtou se další dva bity, které jsou převedeny na celé číslo, a dekódování je ukončeno. Pokud je bit 1, načtou se ještě dva bity, které jsou převedeny na celé číslo M s nejvyšším bitem 1, a načte se další bit. Pokud je 0, je dekódování ukončeno, jinak je zavolána metoda recursiveDecode, která načte dalších $M - 1$ bitů a převede je na celé číslo M, takto to pokračuje voláním metody recursiveDecode dokud není první načtený bit 0.

Příklad 3.4.1
Kódování čísla $n = 32$. Zapíšeme číslo 0. Zakódujeme číslo n kódem beta, $32_{10} = 100000_2$ a $len = 6$. Dále zakódujeme číslo len kódem beta, $6_{10} = 110_2$, $len = 3$ a protože je délka len rovna 3, je kódování ukončeno. Výsledný Even-Rodeh kód je $110|100000|0_2$.

Příklad 3.4.2
Postup při dekódování omega kódu 1101000000_2. Ze streamu načteme první 3 bity, 110_2. Převedeme je na celé číslo, $M = 6$. Načteme dalších M bitů jako celé číslo, $M = 32$. Další skupina bitů by začínala oddělovačem 0, který značí ukončení dekódování. Dekódované číslo je 32.

3.5 Implementace Punctured Elias kódování

Při tomto kódování dochází ke změně pořadí bitů u beta kódování. Vytvořil jsem pomocné pole *binary*, do kterého se prvně bity v normálním pořadí zapíší a následně se z tohoto pole zapíší v obráceném pořadí již do pole *bitArray*.

Kódování P1 probíhá následovně, číslo n je zakódováno kódem beta a poté se obrátí se pořadí bitů. Zapíše se před kód beta číslo 0 a následně je přidáno před kód tolik jedniček, kolik jich obsahuje kód beta.

U dekódování P1 se načítají jedničky do proměnné *CountOne*, dokud není načten bit 0. Poté se načítají bity tak dlouho, dokud není počet načtených jedniček roven *CountOne* a všechny tyto načtené bity se převedou na celé číslo.

Kódování P2 probíhá následovně, číslo n je sníženo o 1, zakódováno kódem beta a poté se obrátí se pořadí bitů. Počet jedniček v kódu beta se zapíše do proměnné *CountOne*. Před obrácený kód beta se přidá číslo 0 a následně je přidáno před kód ještě *CountOne* -1 jedniček.

U dekódování P2 se načítají jedničky do proměnné *CountOne* dokud není načten bit 0. Poté se načítají bity tak dlouho, dokud není počet načtených jedniček roven *CountOne* + 1a všechny tyto načtené bity se převedou na celé číslo. Nakonec se toto číslo zvýší o jedna.

Příklad 3.5.1
P1 Kódování čísla $n = 10$. Zakódujeme číslo n kódem beta, $10_{10} = 1010_2$ a $len = 4$. V dalším kroku obratíme pořadí bitů, 0101_2 a zapíšeme počet jedniček, *countOne* = 2. Přídáme oddělovač 0 a *countOne* jedniček. Výsledný Punsctured Elias P1 kód je $11|0|0101_2$.

Příklad 3.5.2
Postup při dekódování eliasova P1 kódu 11101011_2. Ze streamu načteme tři jedničky a zapíšeme, *countOne* = 3. Přeskočíme oddělovač a načítáme bity, dokud počet načtených jedniček není roven *countOne*, 1011. U načtených bitů změníme pořadí a poté převedeme na celé číslo, $n = 13$.

3.6 Implementace Stoutova R_l kódování

Toto kódování jako jediné ze zpracovaných kódování používá parametr l, který určuje, při jaké délce má být kódování ukončeno.

Kódování probíhá následovně, číslo n je zakódováno kódem beta a přidá se oddělovač 0 před kód beta. Číslo *len* je zakódováno kódem beta a opět přidáno před kód. Poté je zavolána metoda recursiveEncode(), která zjistí, jestli je délka *len* menší nebo rovno parametru l. Pokud ano, je kódování ukončeno a při *len* menší než l jsou přidány před kód nuly do velikosti l. Pokud ne, je délka opět zakódována kódem beta a je znovu volána metoda recursiveEncode(). Takto to pokračuje, dokud není délka *len* rovna nebo menší než l.

U dekódování je načteno v prvním kroku l bitů a ty jsou převedeny na celé číslo n. V dalším kroku je načten jeden bit. Pokud je bit 0, víme, že další skupina n bitů je výsledné číslo. Pokud ne, je volána metoda recursiveDecode(), která načte $n - 1$ bitů a převede na číslo s jedničkovým bitem na začátku. V dalším kroku je načten jeden bit. Pokud je bit 0, víme, že další skupina n bitů je výsledné číslo. Pokud ne, je volána metoda recursiveDecode(). Metoda recursiveDecode() je volána tak dlouho, dokud není nalezen oddělovač 0 a není získáno výsledné číslo.

Příklad 3.6.1
Kódování čísla $n = 18$ pro $l = 4$. Číslo n zakódujeme kódem beta, $18_{10} = 10010_2$ a $len = 5$. V dalším kroku přidáme oddělovač 0 a zakódujeme číslo *len* kódem beta, $5_{10} = 101_2$. Nyní je již číslo *len* menší než číslo l a je přidána před kód jedna nula do velikosti l. Výsledný kód je $0101|0|10010_2$.

Příklad 3.6.2
Postup při dekódování kódu 0101010010_2 pro $l = 4$. Ze streamu načteme l bitů, 0101_2 a převedeme je na celé číslo $M = 5$. Další načtený bit je oddělovač 0, načteme posledních M bitů jako celé číslo $n = 18$.

4. Testování

Testování proběhlo na šesti typech testovacích souborů, každý ve formátu txt. Aby se zamezilo nepřesnostem během měření efektivity, obsahuje každý soubor milion hodnot. Pět souborů nese název data_0_5, data_0_8, data_8_16, data_16_24, data_24_31 podle formule data_x_y a jedná se o soubory s náhodnými daty s uniformním rozložením v rozsahu $< 2^x, 2^y\text{-}1>$. V posledním testovacím souboru data_norm jsou náhodná data s normálním rozložením pravděpodobnosti:

$$f(x) = \frac{1}{s \cdot \sqrt{2 \cdot \pi}} e^{-\frac{(x-m)^2}{2 \cdot s^2}}, \text{ pro } m = 0 \text{ a } s = 128.$$

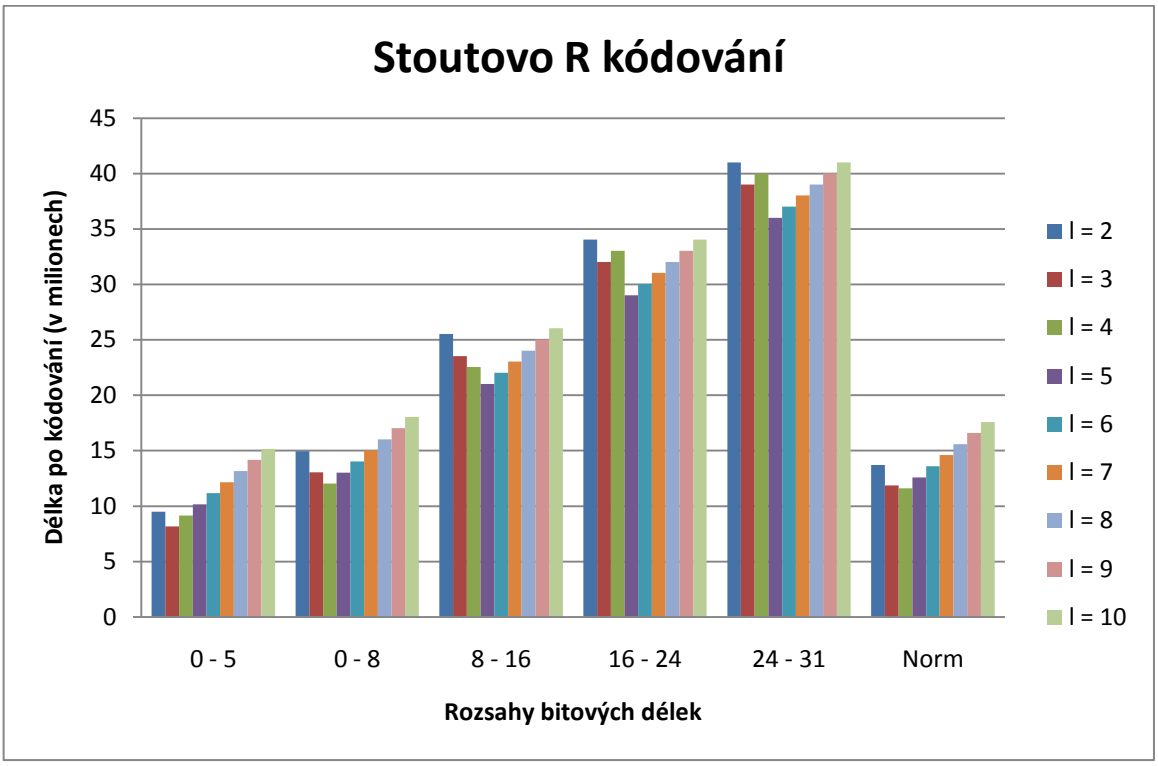
Výsledky byly porovnávány podle výsledných délek z kódování a dle velikosti komprimačního poměru, kde délka původní dat je 32 bitů. Pro testování R_l efektivity Stoutova kódu byl vytvořen samostatný graf, na pro znázornění výběru nejvhodnějšího parametru *l* při porovnávání efektivity mezi různými typy kódování.

4.1 Testování Stoutova R_l kódování

U Stoutova R_l kódování došlo k devíti testováním pro každý rozsah. Testování proběhlo při hodnotě parametru *l* v rozmezí 2 až 10. Nejmenší délky kódování byly dosáhnuty při parametrech l rovných 2, 3 a 4. V tabulce 4.1.1 můžeme vidět výsledky kódování pro jednotlivé parametry *l* a nejlepší hodnota v daném rozsahu byla zvýrazněna tučným písmem.

	Uniformní rozložení (bit)					
Parametr	**0 – 5 bitů**	**0 - 8 bitů**	**8 - 16 bitů**	**16 - 24 bitů**	**24 - 32 bitů**	**normální**
l = 2	9 484 585	14 957 932	25 532 684	34 031 279	41 018 055	13 717 951
l = 3	**8 162 072**	13 040 405	23 532 684	32 031 279	39 018 055	11 867 447
l = 4	9 162 072	**12 031 453**	22 539 432	33 031 279	40 018 055	**11 597 311**
l = 5	10 162 072	13 031 453	**21 030 997**	**29 031 279**	**36 018 055**	12 597 311
l = 6	11 162 072	14 031 453	22 030 997	30 031 279	37 018 055	13 597 311
l = 7	12 162 072	15 031 453	23 030 997	31 031 279	38 018 055	14 597 311
l = 8	13 162 072	16 031 453	24 030 997	32 031 279	39 018 055	15 597 311
l = 9	14 162 072	17 031 453	25 030 997	33 031 279	40 018 055	16 597 311
l = 10	15 162 072	18 031 453	26 030 997	34 031 279	41 018 055	17 597 311

Tabulka 4.1.1: Stoutovo kódování pro různou hodnotu l



Graf 4.1.1: Stoutovo kódování s různými hodnotami *l*.

4.2 Testování na rozsahu délky 0 až 5 bitů

V daném rozsahu je uniformní rozložení čísel 1 až 31. Testovaná kódování měla v této oblasti nejlepší efektivitu z důvodu, že 32bitová délka, se kterou se výsledky testování porovnávají, má pevnou délku na rozdíl od testovacích kódování, které můžou délku měnit a nemusí tedy zanechávat hodnotu čísla v plném rozsahu 32bitů.

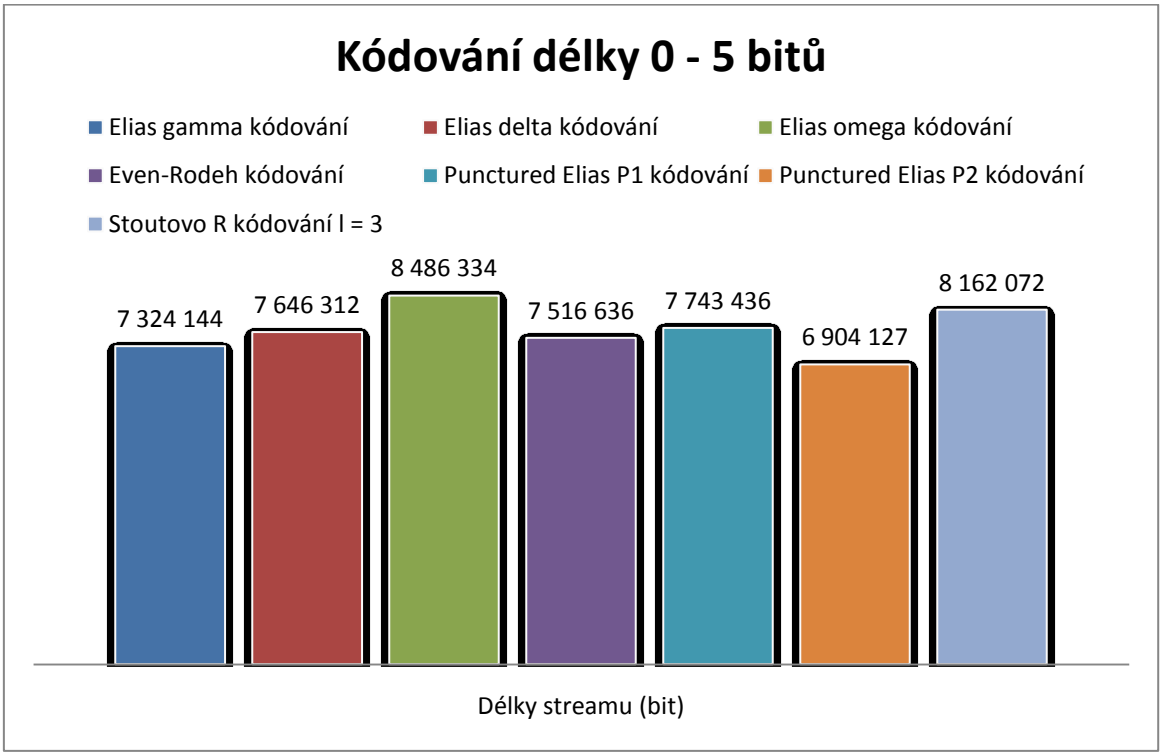
Při pohledu na tabulku 4.2.1, je patrné, že nejefektivnější kódování pro tento rozsah je Punctured Elias P2 kódování, které dosahuje komprimačního poměru 21,58% a úspora místa se dostala na 78,42%.

Velmi dobrých výsledků dosáhlo i Eliasovo gama kódování, které spoléhá na výskyt malých čísel a komprimační poměr se dostal na 22,89%.

Většinou se efektivity jednotlivých kódování od sebe liší o 1% a nejhoršího výsledku dosahuje Eliasovo omega kódování, kde je komprimační poměr 26,52%.

Kódování	Délka streamu [bit]	Komprimační poměr
Eliasovo gama kódování	7 324 144	22,89%
Eliasovo delta kódování	7 646 312	23,89%
Eliasovo omega kódování	8 486 334	26,52%
Even-Rodeh kódování	7 516 636	23,49%
Punctured Elias P1 kódování	7 743 436	24,20%
Punctured Elias P2 kódování	6 904 127	21,58%
Stoutovo R_l kódování $l = 3$	8 162 072	25,21%

Tabulka 4.2.1: Komprimační poměr a délka streamu pro rozsah délky 0 - 5 bitů



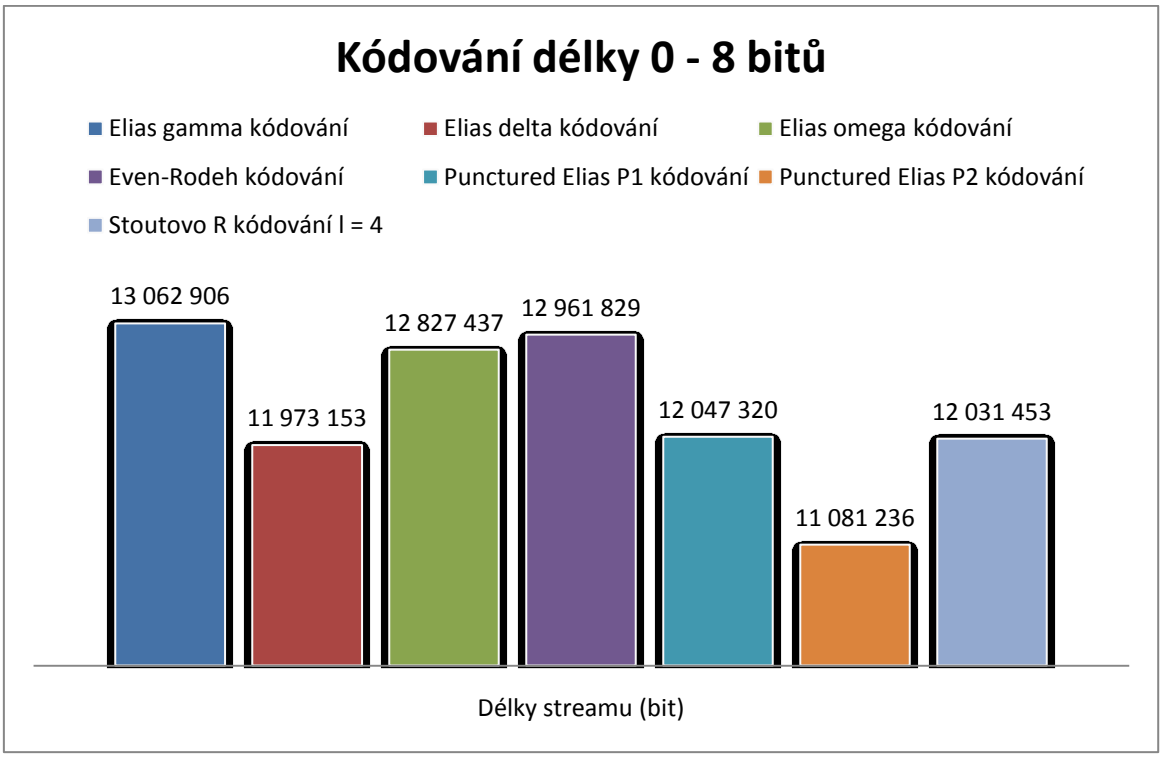
Graf 4.2.1: Délka streamu pro rozsah délky 0 - 5 bitů

4.3 Testování na rozsahu délky 0 až 8 bitů

Testování proběhlo na číslech 1 až 255. S rostoucí délkou a zvyšováním hodnot rozsahu se rozdíl v komprimačním poměru mezi nejlepším a nejhorším kódováním zvětšuje a úspora místa klesá. U předchozího rozsahu je rozdíl mezi jednotlivými typy kódování 4% a zde již 6%. Při porovnání tabulek 4.2.1 a 4.3.1 je patrné, které typy kódování jsou určeny pro velmi malé hodnoty, a jejich komprimační poměr je při malých hodnotách větší, ale pokud se oblast čísel rozšíří, pak tyto kódování nabývají horších hodnot u komprimačního poměru. Jedná se především o Eliasovo gama kódování,

Kódování	Délka streamu [bit]	Komprimační poměr
Eliasovo gama kódování	13 062 906	40,82%
Eliasovo delta kódování	11 973 153	37,42%
Eliasovo omega kódování	12 827 437	40,09%
Even-Rodeh kódování	12 961 829	40,51%
Punctured Elias P1 kódování	12 047 320	37,65%
Punctured Elias P2 kódování	11 081 236	34,63%
Stoutovo R_l kódování $l = 4$	12 031 453	37,60%

Tabulka 4.3.1: Komprimační poměr a délka streamu pro rozsah délky 0 - 8 bitů



Graf 4.3.1: Délka streamu pro rozsah délky 0 - 8 bitů

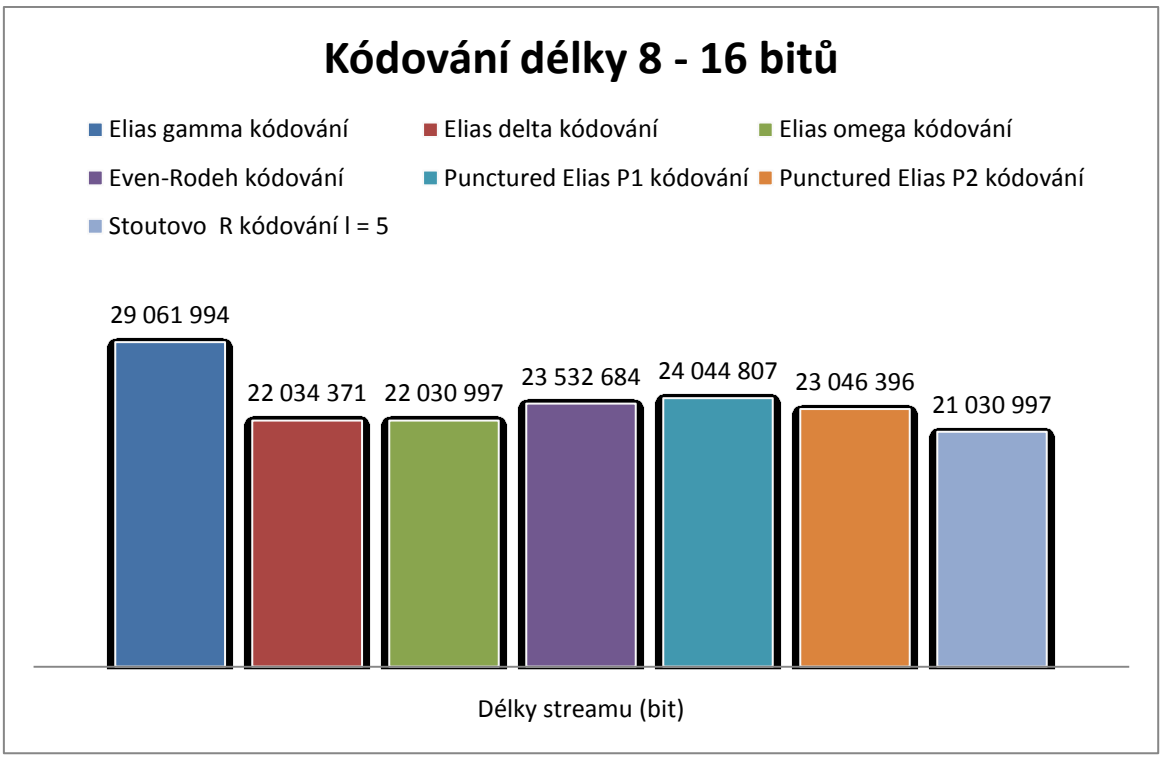
které působí nejefektivněji na poli malých čísel a u předchozího testování je na druhém místě. Zde však skončilo na posledním místě a nejlepšího výsledku pro tento rozsah je dosaženo Punctured Elias P2 kódováním s úsporou 65,37%.

4.4 Testování na rozsahu délky 8 až 16 bitů

Testování čísel je v rozsahu 256 až 65 535 a dosahuje se stále kladného komprimačního poměru, avšak Eliasovo gama kódování se blíží čím dál více k délce 32bitů a rozdíl mezi ním a předposledním typem kódování je 15%, což je velký rozdíl a při porovnání s nejlepším výsledkem je tento rozdíl už jen 25%.

Při pohledu na Graf 4.4.1 je patrné, že nejlepšího výsledku v tomto rozsahu dosáhl Stoutův R_l kód při hodnotě parametru $l = 5$, kde je komprimační poměr 65,72% a úspora místa 34,28%.

Eliasovo kódování delta a omega má téměř totožné hodnoty z testování a liší se jenom o 0,01%. Even-Rodeh kódování se drží prostředních míst efektivity při porovnání s výsledky ostatních typu kódování.



Graf 4.4.1: Délka streamu pro rozsah délky 8 - 16 bitů

Kódování	Délka streamu [bit]	Komprimační poměr
Eliasovo gama kódování	29 061 994	90,82%
Eliasovo delta kódování	22 034 371	68,86%
Eliasovo omega kódování	22 030 997	68,85%
Even-Rodeh kódování	23 532 684	73,54%
Punctured Elias P1 kódování	24 044 807	75,14%
Punctured Elias P2 kódování	23 046 396	72,02%
Stoutovo R_l kódování $l = 5$	21 030 997	65,72%

Tabulka 4.4.1: Komprimační poměr a délka streamu pro rozsah délky 8 - 16 bitů

4.5 Testování na rozsahu délky 16 až 24 bitů

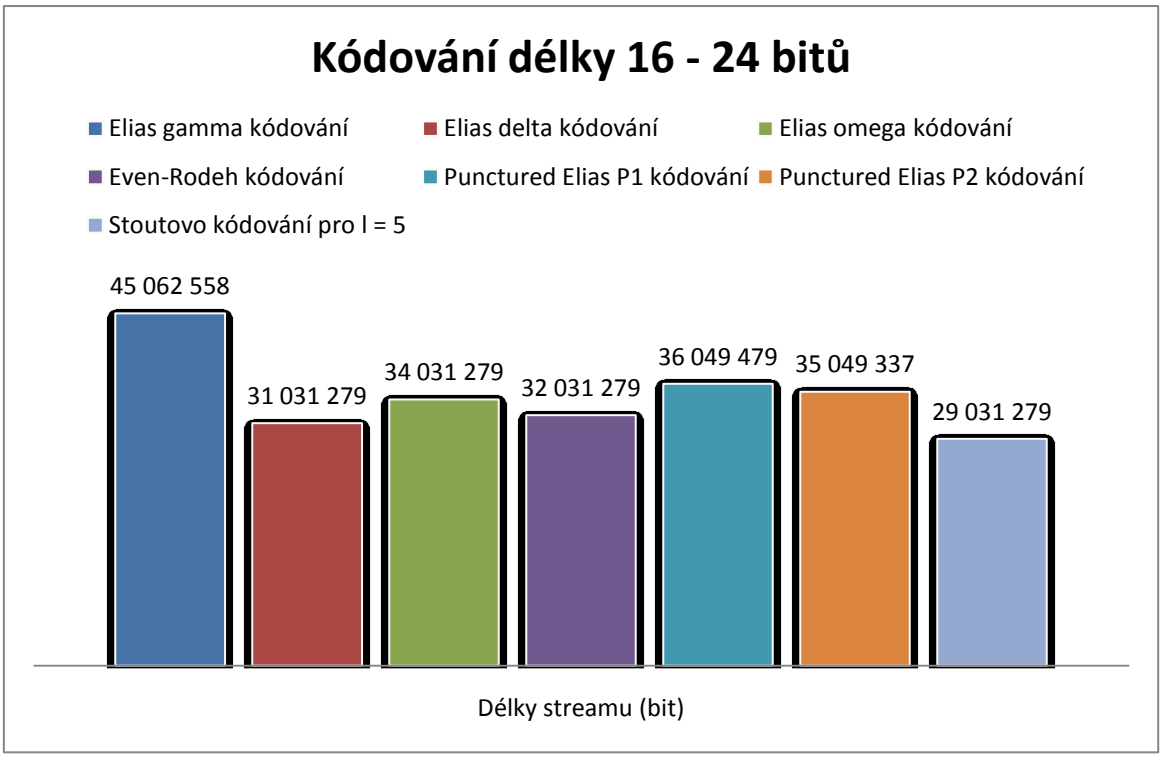
V daném rozsahu je uniformní rozložení čísel 65 536 až 16 777 215 a tyto čísla již nejsou tak často užívána.

Eliasovo kódování je opět nejhorší ze všech testovaných kódování s komprimačním poměrem 140,82% což znamená, že toto kódování se již neoplatí.

Pro tuto oblast čísel je nejefektivnější Stoutovo R_l kódování s parametrem $l = 5$. U tohoto kódování je úspora místa sice jen 9,285% ale při porovnání s ostatními kódování je to jedno z mála kódování, které nezvětšuje velikost testovacího souboru.

Kódování	Délka streamu [bit]	Komprimační poměr
Eliasovo gama kódování	45 062 558	140,82%
Eliasovo delta kódování	31 031 279	96,97%
Eliasovo omega kódování	34 031 279	106,35%
Even-Rodeh kódování	32 031 279	100,10%
Punctured Elias P1 kódování	36 049 479	112,65%
Punctured Elias P2 kódování	35 049 337	109,53%
Stoutovo R_l kódování $l = 5$	29 031 279	90,72%

Tabulka 4.5.1: Komprimační poměr a délka streamu pro rozsah délky 16 - 24 bitů



Graf 4.5.1: Délka streamu pro rozsah délky 16 - 24 bitů

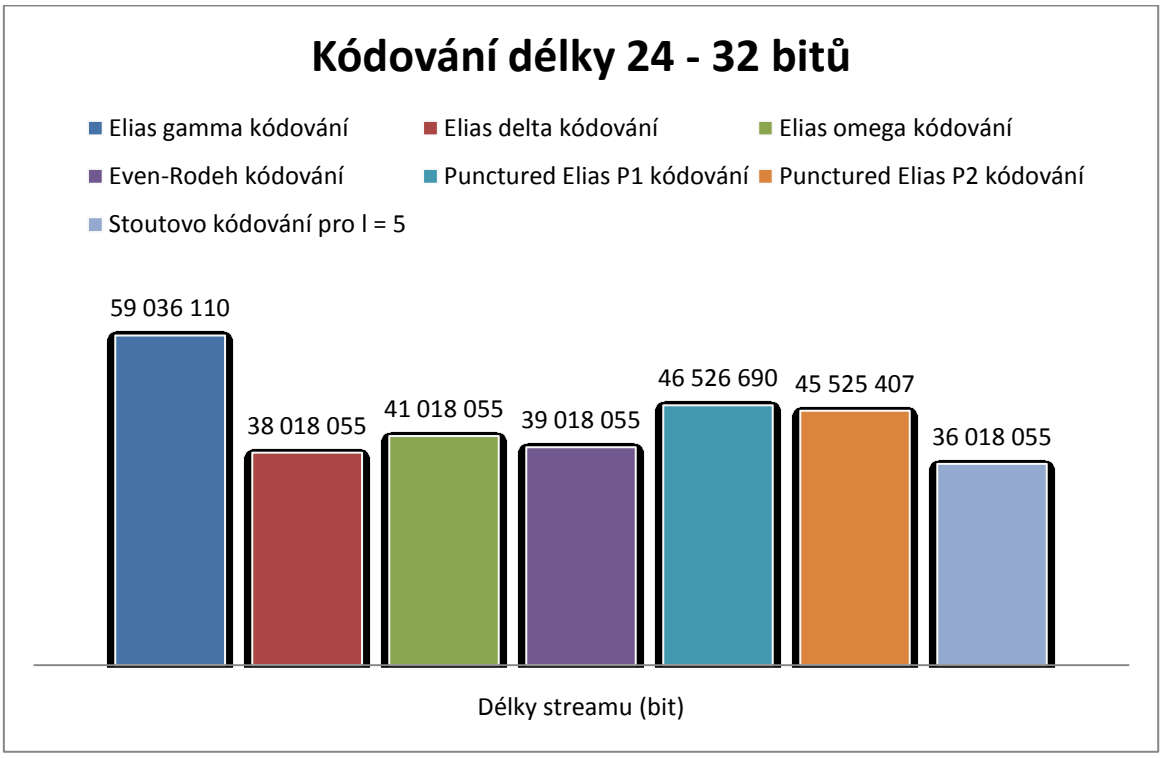
4.6 Testování na rozsahu délky 24 až 32 bitů

Testování proběhlo na málo užívaných číslech v rozsahu 16 777 216 až 2 147 483 647. Všechny typy kódování přesáhly původní velikost testovacího souboru. Rozdíly mezi jednotlivými kódováními jsou od 3% do 30% a nejhoršího výsledku opět dosáhlo Eliasovo gama kódování, kde komprimační poměr dosáhl 148,49%, což je o polovinu delší kód než klasický 32bitový zápis.

Z grafu 4.6.1 můžeme vidět, že nejlepšího komprimačního poměru dosahuje stále Stoutovo R_l kódování, které dosahuje komprimačního poměru v tomto rozsahu 112,56% a v těsném závěsu za ním je Eliasovo delta kódování s komprimačním poměrem 118,81%.

Kódování	Délka streamu [bit]	Komprimační poměr
Eliasovo gama kódování	59 036 110	148,49%
Eliasovo delta kódování	38 018 055	118,81%
Eliasovo omega kódování	41 018 055	128,18%
Even-Rodeh kódování	39 018 055	121,93%
Punctured Elias P1 kódování	46 526 690	145,40%
Punctured Elias P2 kódování	45 525 407	142,27%
Stoutovo R_l kódování $l = 5$	36 018 055	112,56%

Tabulka 4.6.1: Komprimační poměr a délka streamu pro rozsah délky 24 - 32 bitů

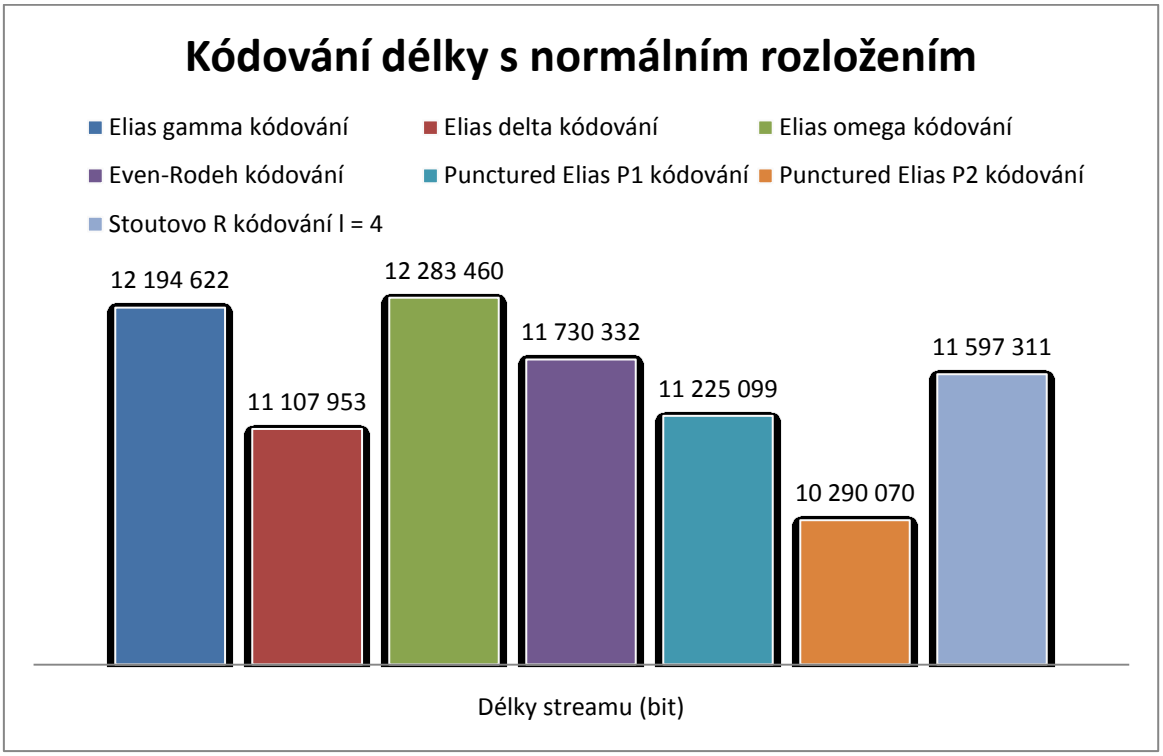


Graf 4.6.1: Délky streamu pro rozsah délky 24 - 32 bitů

4.7 Testování na normálním rozložení

Testování se zde týká číslic v rozmezí jednotek až stovek, kde pravděpodobnost výskytu těchto číslic je daná Gaussovým rozdělením pravděpodobnosti. Z tabulky 4.7.1 můžeme zjistit, že všechny typy kódování jsou pro tento rozsah komprimační a dosahují dobré úrovně komprimace v rozmezí od 32% do 39%.

Jak je vidět v grafu 4.7.1, Punctured Elias P2 kód dosáhl nejlepšího komprimačního poměru 32,16% s úsporou 67,84%. Nejhoršího komprimačního poměru dosáhl Eliasův omega kód.



Graf 4.7.1: Délka streamu pro normální rozložení

Kódování	Délka streamu [bit]	Komprimační poměr
Elias gama kódování	12 194 622	38,11%
Elias delta kódování	11 107 953	34,71%
Elias omega kódování	12 283 460	38,39%
Even-Rodeh kódování	11 730 332	36,66%
Punctured Elias P1 kódování	11 225 099	35,08%
Punctured Elias P2 kódování	10 290 070	32,16%
Stoutovo R_l kódování $l = 4$	11 597 311	36,24%

Tabulka 4.7.1: Komprimační poměr a délka streamu pro normální rozložení.

5. Závěr

V této práci jsem se seznámil s různými způsoby prefixových kódování celých čísel a získané poznatky jsem využil při tvorbě tříd potřebných pro nasazení do kompresního frameworku. Otestoval jsem u Stoutova R_l kódování do hodnoty $l = 10$, abych názorně ukázal, že Stoutovo R_l kódování je nejvýhodnější při hodnotě parametru l v rozsahu < 3, 5>.

U Eliasova gama kódování došlo k velkému výkyvu délky při zvýšení rozsahu čísel. Toto kódování se ověřilo jako nejlepší pro velmi malé hodnoty. Avšak pokud jsou čísla větší, je kódování naopak nejhorší a mnohdy je rozdíl mezi ním a druhým nejhorším kódováním o 30%. Eliasův delta kód se ukázal velmi dobrým kódováním pro všechny rozsahy, kde se většinou umístil na druhém místě. Poslední typ kódování Eliasův omega kód se pro velmi malá čísla jeví jako nejhorší ze všech testovacích kódování, pokud ale čísla mají větší hodnotu, dosahuje toto kódování středních hodnot.

Even-Rodeh kódování nevynikalo v žádném rozsahu, avšak ani nedosahoval nejhorších výsledků. Toto kódování se drželo zlatého středu a osvědčilo se jako dobré kódování pro všechny rozsahy.

V případě Punctured Eliasových kódování, dosahovalo kódování P2 nejlepších výsledků u čísel s rozsahy délek 0 – 16 bitů, naopak při délkách 16 – 24bitů patří toto kódování mezi nejhorší z testovaných. Co se týče kódování P1 bylo vždy horší než P2 a to nejméně o 3% u každého rozsahu. Je to dáno hlavně tím, že kódování P2 je odvozeno z P1 a pracuje s hodnotou o jedna nižší než P1, a také má jeho prefix vždy o jednu jedničku méně.

Stoutovo R_l kódování dosahovalo různých výsledků při různé hodnotě parametru l. Pro velmi malé hodnoty se osvědčila hodnota parametru $l = 3$. Pro rozsah 0 – 8 bitů je nejlepší hodnota $l = 4$. A u velkých čísel řádově dlouhé 8 – 32 bitů měl nejlepší komprimační poměr právě Stoutův R_l kód s hodnotou $l = 5$. Můžeme tedy říci, že dané kódování velmi efektivní pro všechny rozsahy, musíme ale zvolit správně parametr l.

Na závěr bych jenom dodal, že pokud bychom chtěli kódovat vysoká čísla je lepší je ponechat v původním tvaru a nijak je nekódovat. Pro malá čísla je nejefektivnější Punctured Eliasův kód P2. Pro střední a velkou velikost čísel je nejefektivnější Stoutův R_l kód a to s hodnotou parametru l roven 4 a 5.

Literatura

[1] Salomon, David, *Variable-length codes for Data Compression*, 2007
[2] Salomon, David, *Data Compression: The Complete Reference 4. ed.,* 2007

Seznam příloh

K této práci je přiloženo medium CD – ROM, které obsahuje dva adresáře:

1. **Software** – implementované moduly + testovací data
2. **Text** - text bakalářské práce v elektronické podobě ve formátu Adobe Acrobat Reader